Installation Guide

c-tree Plus™

c-tree Plus™ File Handler

Installation Guide

For c-tree Plus Version 6.0

© 1990 FairCom Corporation

ALL RIGHTS RESERVED

Published by
FairCom Corporation
4006 West Broadway
Columbia, MO 65203
USA

© 1990 FairCom Corporation

All rights reserved. No part of this publication may be stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of FairCom Corporation. Printed in the United States of America.

First Printing: November, 1990

c-tree, r-tree, the circular disk logo, and FairCom are registered trademarks of the FairCom Corporation. d-tree and c-tree Plus are trademarks of the FairCom Corporation.

Unix is a trademark of AT & T. MS-DOS, OS/2 and Xenix are trademarks of Microsoft Corporation.

Installation Guide Table of Contents

Source Code Installation 1

Utility Programs 10

Sample Programs 14

Documentation Update 17

Conversion of c-tree V4.3 Applications 24

c-tree Plus™ Installation Guide

The purpose of this Installation Guide is to provide information on

- how to install the **c-tree Plus** source code on your hard disk, and create the application library, **c-tree Plus** utilities, and sample programs
- how to use the **c-tree Plus** utilities
- explanation of the sample programs
- updates to the documentation
- how to convert your **c-tree®** V4.3 applications to **c-tree Plus**

Source Code Installation

This section deals with moving the **c-tree Plus** source from your distribution floppy diskettes to your hard disk, and how to compile and link the **c-tree Plus** code. You should have either four 5¼ inch 360K distribution floppies or two 3½ inch 720K distribution floppies. These diskettes support the following combination of operating systems and compilers:

Operating System	Compiler
MS/PC-DOS	Microsoft C 5.1/6.0 Turbo C/C++ Watcom C
OS/2	Microsoft C 5.1/6.0
Windows 3.0	Microsoft C 6.0 (Windows SDK required)
Unix 386	cc
Xenix 386	cc
Xenix 286	cc
Unix	cc

c-tree Plus offers several modes of operation, and various compile time configuration options. The file ctoptn.h controls these compile time options. For now, we suggest that you accept the configuration as distributed to you. The section "Modifying Configuration Parameters" discusses how you can change your **c-tree Plus** settings.

DOS, OS/2 and Windows Installation

Copying The Code

These three operating system environments all install in the same manner. (OS/2 V1.1 and V1.2 are both supported and behave the same with respect to **c-tree Plus**.) Place the first **c-tree Plus** distribution diskette in the floppy drive of your choice. Enter the command

```
w:ctree
```

where **w** is the floppy disk drive (e.g., b:ctree). You will see a screen similar to

```
USAGE: ctree {Hard Drive} {c-tree Directory} {Floppy Drive} {Compiler}

      where {Hard Drive} = C or D or E or ... (destination)

            {c-tree Directory} = Hard disk directory for c-tree PLUS

            {Floppy Drive} = A or B or ... (source), and

            {Compiler} = MSC51 OS251 MSC60 OS260 WND3 TC WC UNIX

      MSCxx is Microsoft C for DOS. OS2xx is Microsoft C for OS/2

      WND3 is Microsoft C 6.0 for Windows 3.0

      TC is Turbo C and Turbo C++

      WC is Watcom C

      UNIX is generic Unix code for porting to a Unix machine

      Unix386 and Xenix have special installs. See Installation Guide.
```

```
e.g., a:ctree   C   ctree    a   MSC60

      b:ctree   d   \ctp     B   TC
```

To copy the source code, enter a command of the form

```
w:ctree DestinationDrive DestinationDirectory w Compiler
```

where **w** is the floppy drive, and **DestinationDrive** is the hard disk drive on which the **c-tree Plus** code is to be installed. The **DestinationDirectory** will be made relative to the root if it begins with a '\' (back slash), or it will be relative to the current working directory if it doesn't. Do not use a compound directory (i.e., do not use a

directory of the form abc\def). **Compiler** should be selected from the list shown on the screen. Remember to put the **Compiler** entry in **UPPER CASE**.

Here are some sample commands (where the information to the left of the '>' is simply the current drive and current working directory for the drive):

```
B:\>ctree e \ct b TC
```

The above command places the code on drive E in the directory \ct and sets up for Turbo C or Turbo C++. The **c-tree Plus** distribution disks will be read from drive B.

```
C:\source>a:ctree C c_tree a MSC60
```

The above command places the code on drive C in the directory c_tree relative to drive C's current working directory (\source\c_tree) and sets up for Microsoft C 6.0. The diskettes will be read from drive A.

Once started, the **ctree** batch program invoked as shown above will copy all the source code, prompting you when necessary to insert the next distribution diskette. When completed, the file copying process will have actually created two directories: the directory specified on the ctree command line, and a subdirectory with the same name as the compiler code used on the ctree command line.

The first directory will contain the system independent source code. The subdirectory will contain the system specific code, headers and make files.

Using the second example shown above, the system independent code will be placed in the directory \source\c_tree and the Microsoft C 6.0, DOS dependent code, headers and make files will be placed in the directory \source\c_tree\msc60.

Please examine the system independent read.me file in the main **c-tree Plus** directory. Also, the system dependent subdirectory may have a read.me file with important information.

Compiling The Code

Before attempting to compile the **c-tree Plus** source code, make sure that your PATH, LIB, and INCLUDE environment variables are set correctly for your compiler.

In order to compile the source code, make the **c-tree Plus** application library, and create the **c-tree Plus** utilities and sample programs, set the current working directory to the c-tree system dependent subdirectory (\source\c_tree\msc60 in our example). Then enter a command of the form

```
makeapp L
```

The L indicates the large memory model. The other memory models are specified by the letters S for small, M for medium and C for compact. There are two exceptions to the **makeapp L** command.

First, with Turbo C or C++ there are two additional parameters which specify where to find the Turbo C tools. Running **makeapp L** under Turbo C will show you how to properly formulate this make command.

Second, with Windows 3.0, there is no memory model specification: simply enter **makeapp**. The Windows requirements have already been specified. The Windows make only makes the special windows sample program CTDEMO and the **c-tree Plus** DLL. The **c-tree Plus** utilities are designed as character-based applications, and are not supported by the DLL. You can also install the MSC60 code, and use its make files to construct the **c-tree Plus** utilities. When the windows makeapp has successfully completed, you can run the demo program by entering **win ctdemo**.

You can select another memory model, but we suggest beginning with the large memory model since it accommodates all the utilities and sample programs. If you wish to use a different memory model, see the section "Modifying Configuration Parameters" below.

Except for Windows 3.0 (WND3) and Microsoft C 5.1 (MSC51 and OS251), you can also add a target to the make command line which limits the make process to the target file. For example,

```
makeapp L ctixmg.exe
```

will cause the executable sample program CTIXMG to be compiled and linked along with all the necessary supporting modules including the **c-tree Plus** application library. The advantage of adding the target file name is that not all executables will be created and/or brought up to date, saving you time.

Trouble Shooting

The Microsoft 6.0 make program (nmake.exe) may run out of heap space or resource space before finishing all the makes. If this occurs, simply restart the make file. It will continue where it left off, and should finish without further difficulty.

If the creation of a linker/librarian response file fails (some of the makes echo commands to a response file and then invoke the linker/librarian with the file), be sure that your PATH environment variable does not cause a program named echo (echo.com or echo.exe) to be used instead of the DOS/OS2 intrinsic command.

Linking Your Own Applications

Once you have completed the execution of the **c-tree Plus** make file, you can link your own applications to the **c-tree Plus** library. The **c-tree Plus** application library will be named CTALIB.LIB and will be found in a subdirectory of your system dependent subdirectory. Continuing the example begun above, the **c-tree Plus** application library will be found in

```
\source\c_tree\msc60\lapp
```

The final subdirectory is named nAPP where n is L for large, M for medium, C for compact or S for small memory model. You may move the library, but keeping it where it is created will cause less confusion if you use additional compilers or memory models.

You may wish to add your own applications to the make files we have provided.

Windows 3.0 (WND3) is an exception to the above discussion. Its library and DLL are named CTPWIN.LIB and CTPWIN.DLL. They are found in the WND3 subdirectory. The read.me provided in that subdirectory provides more details for the Windows environment.

Unix386, Xenix386 and Xenix286 Installation

Copying The Code

These three operating system environments support a **doscp** utility which makes it easy to copy code from DOS diskettes to your file system. First, create a directory to hold your **c-tree Plus** source code. Make this the current directory. Type **pwd** to be sure. For example:

```
# pwd
/usr
# mkdir ctree_plus
# cd ctree_plus
# pwd
/usr/ctree_plus
```

Second, place the 1st **c-tree Plus** distribution diskette in the desired floppy drive. Enter the command **doscp w:/xyz.mv .** where **w** species the floppy drive and **xyz** is one of unix386, xenix386 or xenix286. This command copies a shell script which will control the rest of the installation. For example,

```
# doscp a:/xenix386.mv .
```

If the **doscp** command cannot access the floppy, it may be necessary to replace the simple drive designation with a device specification. For example,

```
# doscp /dev/fd048ds9:/xenix286.mv .
```

If this is the case, then use this device specification in the next step as well.

Third, enter the command **sh xyz.mv w** where **w** and **xyz** are defined as before. This command will perform the actual copying, prompting you when it is necessary to insert the next distribution diskette. Continuing with the Xenix 386 example:

```
sh xenix386.mv a
```

Fourth, after successful file copying, examine the contents of the read.me files which have been copied over. read.me is the system independent file, and read.me.xyz is the system dependent file (where **xyz** is defined as before).

Compiling The Code

Make files and shell scripts to launch the make files are provided. On 386 systems enter the command **makefpg** and on 286 systems enter the command **makefpg L** to make the non-server, multi-user version of the sample programs. On some systems, it may be necessary to invoke the shell script with an explicit path. For example,

```
./makefpg
```

Please note that the shell scripts which launch the make files will automatically copy the appropriate contents into the ctoptn.h configuration parameter file.

You may also add a target file to the make command line. This will cause the make file to create only the target, and any of the required supporting modules. For example, on a Xenix 286 system,

```
./makefpg L ctixmg.f
```

will cause the executable ctixmg.f to be created or brought up to date, using the large memory model; whereas,

```
./makefpg L
```

will cause all executables in the make file (makefpg.mak) to be brought up to date. The small and medium models can be selected on Xenix 286 systems by substituting S or M for L when the make is launched. We recommend using the large memory model, especially for the **c-tree Plus** utilities. If you wish to use a different memory model, see the section "Modifying Configuration Parameters" below.

Notice that the multi-user executables have a **.f** appended to the file name.

Single user versions of the examples and all the **c-tree Plus** utilities will be created (or brought up to date) with the makesgl shell script. On a 386 system, the command

```
./makesgl ctrbld.s
```

will cause a single-user application library to be built along with the rebuild executable. Note how single-user versions of the programs have a .s appended to the file name. The make file does this automatically.

Linking Your Own Applications

Once you have completed the execution of the **c-tree Plus** make file, you can link your own applications to the **c-tree Plus** library. The **c-tree Plus** application library will be specified during application linking via the **-l** compiler (linker) switch. The single user library will be referenced by

```
-lctpslib
```

on the compiler command line. The multi-user library will be referenced by

```
-lctpflib
```

on the compiler command line. In addition, the **c-tree Plus** library may need routines from an extended library referenced by **-lx** on the command line. For example,

```
cc myprog.c -lctpflib -lx
```

will compile your program and link it with the **c-tree Plus** multi-user library, the extended run-time library and (automatically) with the c run-time library.

Alternatively, you may add your own executables to the make files we have provided. Simply add them to the FILES macro in makefpg.mak and/or makesgl.mak. Then you can make them in the same manner discussed above for the **c-tree Plus** samples and utilities.

Generic Unix Systems

To install **c-tree Plus** on a generic Unix system, use the procedure outlined above for DOS systems, except that you select UNIX for the "compiler" environment. This will perform the file copying on a MS/PC-DOS or OS2 machine. You must then port the code in the **c-tree Plus** system independent directory and the Unix subdirectory to your Unix box. Take some care not to copy the Unix dependent read.me file over the contents of the system independent read.me file, or vice versa.

Once you have ported the code, say by a communications protocol, you then follow the instructions for the Unix 386 system in order to compile the **c-tree Plus** code, create the samples and utilities, and link your own applications to the **c-tree Plus** libraries. Generally, however, you will not use the -lx option during linking in a generic Unix environment. (The make files provided for this environment do not use **-lx** with the link command.)

However, before proceeding with the actual compilation, first perform the following tasks which will check for the appropriate ctoptn.h settings: compile cttest.c with the command

```
cc cttest.c
```

and execute cttest as follows

```
./a.out
```

cttest will report on your system environment and suggest possible changes to the ctoptn.h configuration parameter file.

How To Install Code For An Additional Compiler

If you have already performed the installation procedure, you do not have to repeat the entire file copying procedure if you wish to install the system dependent code for another environment.

First, create an additional subdirectory off of the main **c-tree Plus** system independent directory. For example, suppose you have already installed **c-tree Plus** for Windows 3.0 with the main directory called \ctree, and you want to add the code for Microsoft C 6.0 for DOS. You already have code in the directories \ctree and \ctree\wnd3. Create a new subdirectory called \ctree\msc60.

Second, copy the code from the **c-tree Plus** distribution diskette subdirectory which matches your new environment. With 5¼ inch diskettes, the system dependent code starts on diskette 3. With 3½ inch diskettes, the system dependent code starts on diskette 2. In our example, you would find the MSC60 subdirectory on the distribution diskette and simply copy all of the files found in the MSC60 directory into your newly created \ctree\msc60 directory.

Modifying Configuration Parameters

Each system dependent directory contains a configuration parameter file, named ctoptn.h. The Unix\Xenix systems actually contain several versions of the option file, the appropriate one of which is copied automatically into ctoptn.h at compile time. The configuration file controls the **c-tree Plus** mode of operation (single-user, non-server multi-user, and client/server). It also controls whether various features will be included in the application library. #define constants are used to modify the options.

The program **CTTEST.C** which can be compiled and linked without a c-tree application library, will report possible changes required in ctoptn.h to satisfy machine\compiler specific characteristics.

Mode of Operation

The following combinations of constants contained in ctoptn.h determine the mode of operation as indicated:

single user	**non-server multi-user**	**client/server**
`#define NOTFORCE`	`#define FPUTFGET`	`#define NOTFORCE`
`#define CTBOUND`	`#define CTBOUND`	`#define NO_BOUND`

If you select the non-server multi-user mode of operation, be sure to read Section 9 of the *Programmer's Reference Guide*.

Machine\Compiler Specifics

Use one of the following definitions to specify the word ordering of your machine:

Intel 8086 Least Significant Byte First

```
#define LOW_HIGH
```

Motorola 68000 Most Significant Byte First

```
#define HIGH_LOW
```

Select one of the following three definitions to specify the scanf format requirement for short integers:

%d

```
#define PERC_D
```

%h

```
#define PERC_H
```

%hd

```
#define PERC_HD
```

If possible, use the PROTOTYPE definition. However, if your compiler does not support function prototypes, trying to use #define PROTOTYPE will cause numerous compile error on each and every module.

use prototypes

```
#define PROTOTYPE
```

do not use prototypes

```
#define NO_PROTOTYPE
```

Feature Selection

In the following note that resources and superfiles require variable length data processing.

Feature	Support Feature	Disable Feature
Resources	#define RESOURCE	#define NO_RESOURCE
Superfiles	#define CTSUPER	#define NO_SUPER
Batched Retrievals	#define CTBATCH	#define NO_BATCH
Variable Length Data	#define VARLDATA	#define NO_VARLD
Key Compression	#define VARLKEYS	#define NO_VARLK
ISAM Parameter Files	#define PARMFILE	#define NO_PARMF
r-tree® symbolic names	#define RTREE	#define NO_RTREE
ISAM processing	#define CTS_ISAM	#define NO_ISAM

Small Memory Model

On Intel machines using memory model dependent compilation, you will have to disable resources, superfiles and batched retrievals to stay within the 64K limitation for your code, the **c-tree Plus** library and the c run-time library.

The rebuild utility and any programs including the rebuild capabilities should not be compiled in the small model. It is likely that the code will not fit, and reasonable data buffers are required for proper operation.

Client\Server Compatibility

You can ensure that your application is compatible with the FairCom® Server by linking it with a special library that simulates the interface to the Server. If your application attempts to use information which is only available in the non-server mode of operation, it will not successfully link with the special library. The link step will fail with unresolved external references.

All of the environments, except for Windows 3, have a batch file (shell script) called makeclnt which operates in the same manner as makeapp, except that it builds a server interface library instead of a **c-tree Plus** application library. This library is much smaller. Your applications will also be smaller because the main body of the **c-tree Plus** code is not linked to it.

The Servers come with the actual interface library modules which are very similar to the ones built by makeclnt, except that they include the actual communications protocols.

Please note that we have only included the CTIXMG example program in the makeclnt make files. There is no point in creating all of the samples and utilities since the executable which results from makeclnt is not functional. It will only display a message indicating that the library tests for server compatibility. You can add your own applications to the makeclnt.mak files.

In DOS/OS2 systems, the simulated server interface library is named CTXLIB.LIB. In Unix/Xenix systems, the interface library is referenced by **-lctpxlib** on the linker command line.

Utility Programs

The **c-tree Plus** utility programs provide mechanisms to maintain, recreate and reorganize your data and index files. Also, two programs are provided to facilitate porting **c-tree** V4.3 data and index files to the **c-tree Plus** format.

Utility Program Descriptions

If a utility program is run without any command line arguments, it will either display the appropriate command line syntax, or it will prompt you for the necessary information. Entries on the command line surrounded by square braces [ ] are optional. Entries surrounded by angle braces < > indicate that a user supplied value (unquoted string or number) is to be entered. Other entries are literals which should be entered as shown. (However, the literals may be in lower or upper case.)

CTCMPC - file compaction

```
ctcmpc <name of file to compact>
```

This utility is run in single user mode. It takes the name of a fixed or variable length data file or an index file. It creates a compacted copy in the file with name CTREE.TMP.

It does not support indexes contained in superfiles or index files which support transaction processing. To compact an index file which is a member of a superfile and/or which supports transactions, use the utility CTDIDX (which can be run with a server) to create a flat key file. Then use the utility CTINDX which will rebuild the index compactly.

CTCV43 - convert V4.3 data file to c-tree Plus format

```
ctcv43 <V4.3 data file name> <c-tree Plus file name>
```

This utility will convert a data file from **c-tree** V4.3 to the **c-tree Plus** format. If the new data file does not exist, it will be created. CTCV43 is designed to run in single user mode. It supports both fixed and variable length files.

After converting the data files, you may use the rebuild utility CTRBLD to construct the associated index files. If you have stand-alone indices to convert, use CTIN43 to first create a flat key file, and then use CTINDX to build the new index file. Note that CTIN43, provided with **c-tree Plus**, must be linked with your **c-tree** V4.3 application library.

CTDIDX - create flat key file

```
ctdidx <index file name> <index member #> <flat key file name>
```

This utility will run either in single user mode or with a FairCom Server. The server mode is necessary to dump the keys from an index which supports transaction processing.

CTDIDX creates a flat file which contains each key value (at full, uncompressed length) followed by the associated record position in long integer (4-byte) form.

CTDIDX reads each key (in order) from the index member designated (0 for the host index, 1 for the first member, etc) and places the keys in the flat file it creates.

CTEXMC - file creation utility / example

```
ctexmc <ParameterFileName> [U<UserId> P<UserPassWord>
        F<FilePassWord>]");
```

This utility can run with or without a server. It creates the files specified in the named parameter file. If any of the files already exist, it will terminate unsuccessfully. The UserId and password parameters are optional, and only apply to server environments.

CTFLAT - create c-tree Plus file from a flat file

```
ctflat <flat file> <new file> <record length>
        [<file mode> <ext size>]
```

CTFLAT creates a **c-tree Plus** fixed length data file from an existing flat file with the specified record length. Optionally, you can specify the file mode to use when creating the new file (e.g., to enable transaction processing), as well as the file size extension parameter.

CTFLAT assumes that the flat file begins at offset zero and has the record length indicated on the command line. You can modify the source to make it more appropriate to your specific circumstances.

CTINDX - create index file from flat key file.

```
ctindx <flat key file> <index file> <index member>
        [<transformation file>]
```

This utility runs in single user mode (but it does support indices which require transaction processing). The index file must already exist, and the index member (0 for the host index, 1 for the first additional index member, etc.) must be empty. THE KEY LENGTH OF THE INDEX MEMBER IS USED TO DEDUCE THE KEY LENGTH IN THE FLAT FILE. You can optionally specify a transformation file which will specify how to transform each key value before entry into the index file.

The optional transformation file is a text file similar to a ISAM parameter file formed as follows:

```
<# of segments>
   <offset> <length> <mode>
  ... repeated for each segment ...
```

where an offset of zero corresponds to the first byte of the key value stored in the flat key file, the length dictates how many bytes are affected by the transformation,

and mode is interpreted as shown in the "Key Segment Modes" section of Chapter 5 of the *Programmer's Reference Guide*.

CTIN43 - create flat key file from V4.3 index file

```
ctin43 <index file> <index member #> <flat key file>
```

CTIN43 MUST BE COMPILED AND LINKED WITH A **c-tree** V4.3 APPLICATION LIBRARY. It will not work with a **c-tree Plus** application library. CTIN43 runs in single user mode, and dumps the keys in order (for the designated index member) to a flat file. It uses the same form that CTDIDX and CTINDX use. Therefore, you can convert an index from V4.3 to **c-tree Plus** by first using CTIN43 and then CTINDX.

CTPKEY - print key utility

```
ctpkey <parameter file name> [<# of entries>]
```

CTPKEY prints out 1 (or more if the optional # of entries is specified) key value from each index file in the parameter file, showing how each key segment is defined. The keys are displayed in hex and ASCII. This utility will run in all modes of operation. It is useful if you are having trouble retrieving data. CTPKEY will allow you to see how a key value is constructed from the data.

By changing a #define, ctpkey.c can be changed from a main line program to a callable function by name of ctpkey with the same parameters as OPNIFILX.

CTRBLD - parameter file rebuild utility

```
ctrbld <parameter file name>
```

CTRBLD must be run in single user mode. It cleans up the data files and completely recreates the index files (in place). If the index file(s) do not exist, they are created.

CTRBLD can only be used on a superfile if the superfile has first been processed by CTSBLD which performs a necessary prepass. When CTSBLD completes, the data file members are in good form, but the index files are empty. CTRBLD (or CTINDX or the RBLIFIL function) can be used to recreate the index file contents.

CTSBLD - superfile rebuild prepass

```
ctsbld <superfile name>
```

CTSBLD processes the superfile in place. It cleans up the data files and empties the index files. The superfile must be processed by either CTRBLD or the RBLIFIL function before it is usable.

CTSFIL - superfile contents

```
ctsfil <superfile name>
```

CTSFIL lists the members of the superfile, indicating whether each member is an index file, fixed length data file, or variable length data file.

CTUNF1 - file reformatting utility

```
ctunf1 <file name> <new alignment> <new flavor>

        alignment    = 1 (byte)
                     = 2 (word)
                     = 4 (double word)

        flavor       = L (LSB first, e.g., 8086 form)
                     = M (MSB first, e.g., 68000 form)
```

CTUNF1 reformats a file IN PLACE provided the new alignment restriction does not cause the data record contents to become misaligned. CTUNF1 takes the place of the UNIFRMAT compile time option; plus it handles data record contents if record definitions have been provided. CTUNF1 is used to port data between different machine architectures.

CTUNF1 uses record definitions, if present, to convert the contents of the data records. A record definition is stored as a resource in a data file using the PUTDODA function. In general, the contents of resource records will NOT be converted. Therefore, if you intend to port your files, you should take care with the resources you directly maintain yourself to ensure they are system independent. However, the resources pertaining to file and record definitions and index alternative collating sequences maintained by FairCom will have their contents automatically converted by CTUNF1.

CTUNF1 runs in single user mode only. It supports data files, index files and superfiles.

Sample Programs

Before running the example programs, be sure that the parameter files supplied with **c-tree Plus** (ctexam.p, ctmark.p, ctvxam.p, isam.p) are in the same directory as the executable sample programs. Otherwise, the samples which require a parameter file will cause an error 102 (could not open parameter file) to occur.

Also, if you have disabled r-tree® Report Generator support (by changing #define RTREE to #define NO_RTREE in ctoptn.h), then you must remove the symbolic names at the end of each data and index line in the parameter files.

On systems which enforce double word alignment for long integers, such as Unix 386, you must add 2 to each of the key segment offsets in the ctexam.p parameter file.

Sample Program Highlights

The sample programs use the same command line format as described in the "Utility Programs" section above.

CTEXMG - ISAM parameter file database example

```
ctexmg
```

CTEXMG uses the parameter file ctexam.p supplied with **c-tree Plus**. The data file is a fixed length record supporting name, number and Zip code indices. CTEXMG does nit use any command line arguments.

You can add new records, scan the records by key, delete records, or update records. You must first use CTEXMC to create the files used by CTEXMG.

This example will run single-user, multi-user non-server or with a FairCom Server. It uses a three buffer technique to determine if two users are updating the same record (without having to hold data record locks during the interactive update).

Before executing CTEXMG, be sure to use CTEXMC to create the files in parameter file ctexam.p. Otherwise an error 12 (could not open file) will be returned.

CTIXMG - Incremental ISAM / Resource example

```
ctixmg [<UserID>  <UserPassWord> <ServerName>]
```

CTIXMG supports variable length data records. It uses Incremental ISAM structures to specify the file parameters for creation. It uses the OPNRFIL function to open the ISAM files based on the file definition automatically stored as a resource in the data file. The optional UserId, UserPassWord and ServerName parameters only apply to server-based systems.

CTIXMG will run in all **c-tree Plus** modes of operation. CTIXMG permits transaction processing control including TRANBEG, TRANSAV, TRANRST, TRANEND and TRANABT. In a non-transaction processing system, these commands will only affect the file locking.

Also, CTIXMG will automatically create the files it needs if they do not exist; and will rebuild the files if they appear corrupt. Of course, in transaction processing environments, automatic recovery replaces the need for rebuilding the files.

CTSXMG - Incremental ISAM / Record Schema example

```
ctsxmg [<UserID>  <UserPassWord> <ServerName>]
```

CTSXMG is based directly on CTIXMG except that a record definition is stored in the data file via PUTDODA. Also, the key segments are described using the segment modes which reference field definitions. The optional UserId, UserPassWord and ServerName parameters only apply to server-based systems.

CTMARK - simulated data generation program

```
ctmark [a<action>] [n<# of trials>] [m<transaction mode>]
            [fc] [r<seed>] [z<parameter file name>]
```

CTMARK uses the parameter file ctmark.p supplied with **c-tree Plus**. The data file is made up of 128-byte fixed length records. Each record is keyed by unique number, name and department code. The data is generated randomly, but has very reasonable characteristics. The names are of varying length with a typical amount of leading character redundancy. The department codes are highly repetitive.

CTMARK runs in all modes of operation.

```
<action>            A - add data
                    G - retrieve data in random order
                    D - delete data (retrieved in random order)
                    U - update data (retrieved in random order)
                    defaults to G

<transaction mode>      N - no transaction processing
                    T - transaction processing with logging
                    t - transaction processing without logging
                    defaults to N

<# of trials>           defaults to 100 if not specified

<seed>              random number seed (0-65535)
                    defaults to 1

fc                  if present, fc indicates that the files in
                    ctmark.p should be created.
                    defaults to no file creation

<parameter file>    ISAM parameter file name
                    defaults to ctmark.p
```

CTVXMG - variable record length ISAM parameter file example

```
ctvxmg
```

CTVXMG supports a simple database similar to CTEXMG except that the data records are variable in length. Use CTEXMC with the parameter file ctvxam.p to first create the files CTVXMG requires. Note how this parameter file incorporates a superfile to hold the data and index.

CTVXMG does not use any command line arguments.

LOWLEVEL ISAM ISAM1 ISAM2 ISAM3

Chapter 7 of the *Programmer's Reference Guide* discusses the introductory sample programs LOWLEVEL, ISAM, ISAM1, ISAM2 and ISAM3. It suffices to say here that each of these programs uses the existence of a command line argument (which may be anything) to indicate that the associated files need to be created.

Documentation Update

This section contains some important additions and corrections to the **c-tree Plus** manuals. Be sure to read this section.

Descriptive Function Names

c-tree Plus permits you to call its functions using either a descriptive name (e.g., **AddResource**) or with a shorthand name (e.g., **ADDRES**). Several of the descriptive names used in the manual have been changed to avoid conflicts with other run-time libraries. In particular, the following changes have been made:

Old Name	New Name
`AddResource`	`AddCtResource`
`DeleteResource`	`DeleteCtResource`
`EnableResource`	`EnableCtResource`
`GetResource`	`GetCtResource`
`UpdateResource`	`UpdateCtResource`
`LockData`	`LockCtData`

The descriptive names are defined in the header file ctlfun.h.

Resource Locks

c-tree Plus assumes that resource entries will be locked only for short intervals. This means you should process resource updates without permitting user interactions to occur during the actual update. Do not lock a resource and then request user input. If necessary, adopt a three buffer approach as described in Section 6 of the

Programmer's Reference Guide to permit a resource update which requires user interaction.

If a file is created with transaction control (either PREIMG or TRNLOG has been OR'ed into the file mode at create time), then one must be careful not to include resource updates (additions, deletions or updates) in long transactions because locks cannot be released until a transaction commits or aborts.

Superfile Restrictions

Node Size

When a superfile is created, the current size of index nodes becomes a critical parameter. (An index node is equivalent to a record of the index file. In non-server systems, the index node size is controlled by the *sect* parameter in calls to INTREE, INTREEX, INTISAM and INTISAMX, or by the third parameter in a ISAM parameter file. In server systems, the node size is controlled by the server configuration; ordinarily the server will be configured with a *sect* parameter of sixteen.)

Whenever the superfile is opened, the index node size currently in effect must be the same as the index node size at the time the superfile was created. This is not usually problem unless one wishes to move the file between different environments. Error SPAG_ERR will result if the node size does not match. (By contrast, an ordinary index file can be opened as long as the current node size is not smaller than the nodes size at the time the index file was created.)

Temporary Files

You should not use superfile members for temporary files. Repeated temporary file creations and deletions will adversely affect the superfile space management.

#include Files

It is not necessary to add the ctifil.h include file in your applications. It will be included automatically.

The SQL application header file ctsql.h is distributed with the SQL Server.

Function Descriptions

Begin (TRANBEG)

trnmod may have **ENABLE** or **ENABLE_BLK** OR'ed in with the type of transaction. **ENABLE_BLK** means that record locks will cause the process to sleep until the lock is available instead of returning with DLOK_ERR (42).

In the RETURN information, error 99 is incorrectly spelled as TTYPE_ERR instead of TTYP_ERR.

FirstKey (FRSKEY)

The shorthand name for **FirstKey**, **FRSKEY**, is omitted from the function description.

GetFileInfo (GETFIL)

In addition to the mode parameters listed in the function description, you may also use the following modes:

mode Constant	GETFIL Return Value
KEYDUP	0 if no duplicates are allowed, 1 otherwise
KEYMEM	index member number (0 for host, 1 first additional member, etc.)
KEYPAD	trailing padding byte (which is compressed with key types 8 and 12)

OpenISAMXtd (OPNISAMX)

There is no *dbufs* parameter. The number of data file buffers is set equal to the number of index file buffers contained in the parameter file. The correct function prototype for **OpenIsamXtd** is

```
COUNT OPNISAMX(pTEXT filnam,COUNT userprof,pTEXT userid,
      pTEXT userword,pTEXT servname,pTEXT fileword);
```

Field Type Changes

The field type chart found on page D-4 of the *Programmer's Reference Guide* should be replaced with the following:

Symbolic Constant	Interpretation
CT_BOOL	one byte Boolean
CT_CHAR	signed char
CT_CHARU	unsigned char
CT_INT2	signed two byte integer
CT_INT2U	unsigned two byte integer
CT_INT4	signed four byte integer
CT_INT4U	unsigned four byte integer

```
CT_MONEY        signed four byte integer interpreted as
                number of pennies
CT_DATE         unsigned four byte integer interpreted as date
CT_TIME         unsigned four byte integer interpreted as time
CT_SFLOAT       four byte floating point
CT_DFLOAT       eight byte floating point
CT_SQLBCD       ten byte scaled BCD (includes exponent)
CT_EFLOAT       extended precision float (not supported as
                a key segment)
CT_TIMES        eight byte extended range date and time structure
CT_ARRAY        arbitrary fixed length data
CT_RESRVD       reserved for future use
CT_FSTRING      fixed length field with delimited data
CT_FPSTRING     fixed length data with one byte length count
CT_F2STRING     fixed length data with two byte length count
CT_F4STRING     fixed length data with four byte length count
CT_STRING       varying length field with delimited data
CT_PSTRING      varying length data with one byte length count
CT_2STRING      varying length data with two byte length count
CT_4STRING      varying length data with four byte length count
```

(The unscaled BCD and BCDMONEY have been eliminated, and a Boolean type and time structure have been added)

The fields which use one or more leading bytes to specify their length include the length byte(s) in the length value.

Key Segment Mode Changes

Key segment mode **BCDSEG** (number 11) is not an active key segment. Segment mode 11 is reserved for future use.

Low Level Key Transformation

c-tree Plus makes it possible to transform key values (or portions of key values) in a manner similar to that used by the ISAM functions even if you are using an index file only with low level functions. The function **cttseg** function transforms a key (or portion of a key) in place and is defined as

```
NINT cttseg(COUNT spos,COUNT mod,COUNT slen,pTEXT target,
            pCOUNT aq,pConvMap mp)
```

where *mod* is the segment mode as described in the "Key Segment Modes" section of Chapter 5 of the *Programmer's Reference Guide*, *slen* is the length of the information to transform, *tarptr* points to the (part of the) key value to be transformed, *aq* points to an optional alternative collating sequence, and *mp* points to a record schema. Either of these last two parameters may be NULL. The *spos* parameter is

only used if the segment mode requires the record schema. In this case, *spos* specifies which field of the schema is used to define the key segment type. **cttseg** returns zero if successful, or a low level error code if not.

For example, if kvp points to a key value comprised of a double float, then the call

```
errc = cttseg(0,FLTSEG,8,(pTEXT) kvp,NULL,NULL);
```

will cause the key value to be transformed in place.

Error Codes

sysiocod

In addition to uerr_cod, isam_err, and sql_err which contain error codes from lowlevel, ISAM and SQL functions, respectively, **c-tree Plus** also sets its global error variable **sysiocod** to the value of errno when an I/O error occurs. errno is the Language C run-time error variable which is automatically maintained by the C run-time library.

Unlike uerr_cod, sysiocod is not reset by new calls to **c-tree Plus**. It is only set to errno if an open, create, seek, read, write or lock function fails.

Error Code Additions

Value	Constant	Explanation
417	SPAG_ERR	superfile created with different index node size
418	SNAM_ERR	superfile created with maximum name length exceeding current maximum
439	DCNV_ERR	file definition conversion missing: call FairCom
462	SHDR_ERR	no superfile directory found during CTSBLD processing
463	UQID_ERR	file does not have unique Server/File number

Error values from 471 to 496, inclusive, are returned by the internal sort routine used during index file rebuild. These error numbers correspond to errors 101 through 126 listed in the **CTSORT.C** module.

Error Code Corrections

Value	Constant	Explanation
53	FTYP_ERR	the file's file mode is inconsistent with the compile time options selected in ctoptn.h
60	KFRC_ERR	*percnt* parameter must be between 0 and 100

```
500   SQLINIT_ERR SQL engine could not be initialized at server
                  start up
501   SQLCONNECT_ERR
                  could not initialize SQL during user logon
502   SQL_REQUEST_ERROR
                  file or memory error during statement processing
```

SQL \ c-tree Plus File Compatibility

In order to ensure that standard **c-tree Plus** files created and manipulated independent of SQL can be used easily with the SQL Server, you should restrict the field types used to long and short integers, single and double precision floats, null terminated character strings, chars, and the two SQL extended types: CT_SQLBCD and CT_TIMES where the structures defining these types are found in ctsql.h.

If the first field in your record contains binary information, then be sure that you place a two byte delete flag field at the very front of the record. In essence, your first field will now become the second field of the record.

If the first field of a record contains character data, then do not place a delete flag field at the front of the record.

When creating SQL tables, be sure to place the variable length columns (VCHAR) at the end of the table to facilitate accessing the information with **c-tree Plus** ISAM and/or low level functions.

SQL \ C Interface Functions

destbuf destfil setdest

The destbuf, destfil, and setdest functions as described on page 15-6 of the *Programmer's Reference Guide* and on pages 3-3, 3-4 and 3-8 of the *Function Reference Guide* are only available with single-user SQL servers. In multi-user systems, the output will always be returned to your program buffers.

Embedded SQL Example

The third from the last line of the embedded SQL example shown on the top of page 15-4 of the *Programmer's Reference Guide* should be

```
EXEC SQL FETCH X INTO :col1, :col2;
```

where the colons immediately precede the names of the local variables.

VARSPEC

The VARSPEC structure discussed with the setvars function on page 3-9 of the *Function Reference Guide* is found in ctsql.h (distributed with the SQL Server) and is defined as

```
typedef struct _varspec
{
        enum HOST_TYPE type;
        int len;
        void *ptr;
        char *pat;
        int isindic;
} VARSPEC;
```

where HOST_TYPE is defined as

```
enum HOST_TYPE
{
        SHORT_INTEGER,
        LONG_INTEGER,
        FLOAT,
        DOUBLE,
        CHAR_ARRAY,
        CHAR_1,
        DATE_TIME,
        DECIMAL,
};
```

sql2

The **sql2** function should be defined as

```
int sql2(statement, nv, vars, ...)
char *statement;
int    nv;
VARSPEC *vars;
```

where there are *nv* instances of the *vars* parameter (instead of *vars* being a pointer to an array of VARPSEC structures). For example, if *nv* is three, then **sql2** would be called with five parameters: *statement*, *nv*, *vars1*, *vars2* and *vars3*. *statement* is either a FETCH, FETCHP or SELECT INTO statement.

Conversion of c-tree V4.3 Applications (Differences from Prior Versions)

The conversion of an existing **c-tree** V4.3 application program involves two main aspects: file conversion and program conversion. The files from **c-tree** V4.3 are not compatible with the **c-tree Plus** file format. Programs may be compatible, but many will require some modification.

Application Conversion Details

data & index files Data files are converted using the utility program CTCV43 provided with **c-tree Plus**. Index files associated with these data files should then be rebuilt using the **c-tree Plus** rebuild utility CTRBLD or with the RBLIFIL function in **c-tree Plus**.

Stand-alone index files are converted with the programs CTIN43 and CTINDX provided with **c-tree Plus**. CTIN43 is linked with your V4.3 library. It produces a flat key file for each index. CTINDX is linked with your **c-tree Plus** library. It creates a **c-tree Plus** index from the flat key file produced by CTIN43.

ISAM Info ISAM Parameter files and Incremental ISAM Structures are compatible.

header info **c-tree Plus** applications cannot directly access the index file headers via the

```
(ct_key+filno)->member
```

reference. Instead, use the GETFIL and GETNAM functions which are part of the **c-tree Plus** library to get the information. For example, the data record length is accessible with the following expression in **c-tree** V4.3

```
(ct_key+filno)->reclen
```

but in **c-tree Plus** the record length is returned by the function call

```
GETFIL(filno,RECLEN).
```

Note that GETFIL returns a long integer since it is also used to return header values such as the file size. All references of the form

```
extern CTFILE *ctnum;
```

must be removed. If you are using such a statement, then you must rely on the GETFIL and GETNAM functions instead.

length parms ALL LENGTH RELATED PARAMETERS HAVE BECOME 4-BYTE VALUES. We now provide function prototypes which should catch (and/or correct) this change. For example, in REDVREC(datno,recptr,bufsiz), bufsiz is now a four byte

quantity.

frmkey If you use the frmkey function, it now takes an additional parameter which specifies the length (as a 4-byte quantity) of the record buffer from which the key is extracted. See the function prototype for frmkey in CTDECL.H.

current ISAM rec cur_recno[ ] and cur_image[ ] no longer exist. Also, it is no longer necessary to use two buffers for record updates (although using two buffers will NOT cause any problem for **c-tree Plus**). **c-tree Plus** maintains the necessary information internally (for each user).

To determine the current ISAM record position, use GETCURP(datno) instead of cur_recno[datno]. If you wish to reset the current ISAM record after a successful rewrite, use UPDCURI(datno,SWTCURI) instead of manipulating cur_recno and cur_image. See the CTVXMG.C code for an example of this technique. See UPDCURI in the *Function Reference Guide* for additional capabilities.

To set the current ISAM record to a specified byte position (and/or record image) use SETCURI(datno,recbyt,recptr,bufsiz) where recbyt is required and specifies the record position. If non-null, recptr points to a record buffer containing an image of the data record. If non-zero, bufsiz (a 4-byte value), specifies the length of the record buffer. If recptr is non-null and bufsiz is passed as 0L, then bufsiz is assumed to be the record length defined for the file at the time the file was created. For variable length files, this corresponds to the fixed length portion of the file.

key segment info If you have accessed key segment information (such as a segment length or segment mode), you can now retrieve it via a call to

```
ctgetseginfo(keyno,segment_no,mode)
```

where segment_no is a zero based segment number, and mode is one of

```
ctSEGPOS    - segment offset
ctSEGMOD    - segment mode
ctSEGLEN    - segment length
```

#include's Be sure to change the c-tree include files to the following:

```
#include "ctstdr.h"
#include "ctoptn.h"
#include "ctaerr.h"
#include "ctdecl.h"
#include "cterrc.h"
```

Note that ctaerr.h includes the definitions for uerr_cod, isam_err and isam_fil. It is not necessary to include ctifil.h (as specified in the manual) since it will be included

automatically.

old key types Key types 1, 2 and 3 have been phased out. They supported integer and floating point keys. You can use appropriate key segments to support all of these types. Change the key type to 0 (unless you desire key compression) and modify your ISAM key segment modes. If your application only uses the low level functions, you can transform the keys as required using the cttseg function described in the "Documentation Update" section of this Installation Guide.

TRANSACTION The TRANSACTION function supported in **c-tree** V4.3 (for purposes of keeping a server from shutting down in the middle of an application procedure) now simply returns without error. It cause no action to be taken.

c-tree Plus supports extensive transaction processing in the server mode via the TRANBEG, TRANABT, TRANEND, TRANSAV and TRANRST functions.

TFRMKEY **c-tree Plus** supports the automatic calling of TFRMKEY to transform target key values used in ISAM level search routines. If you use the traditional c-tree functions to initialize **c-tree Plus** (INTREE, INTISAM, OPNISAM or CREISAM), then this automatic feature is disabled (and your existing application will be unaffected). If you use the extended forms of these functions (INTREEX, INTISAMX, OPNISAMX, CREISAMX), then you must set the userprof parameter to USERPRF_NTKEY to disable the automatic calls to TFRMKEY. Otherwise, your existing calls will be followed by the automatic calls which will cause problems.

UNIFRMAT **c-tree Plus** does NOT support the UNIFRMAT compile time option. Instead, **c-tree Plus** provides a utility program, CTUNF1, which will convert data or index files (including the contents of data records) to a new format.

FPUTONLY There is no longer a FPUTONLY disk I/O protocol. **c-tree Plus** now lets you select on a file-by-file basis whether or not to force updates directly to disk. 'OR' the WRITETHRU constant (64) into the file mode to get the same affect as the FPUTONLY I/O protocol.

DOSFLUSH DOSFLUSH support is being phased-out. As distributed, DOSFLUSH will only become effective in the FPUTFGET (non-server, multi-user) configuration. If necessary, add DOSFLUSH to your CTOPTN.H configuration file. (CTOPT2.H contains preprocessor commands which turn DOSFLUSH off if NOTFORCE is selected.)

Transaction processing provides a much more effective means to ensure the consistency and completeness of the data.

r-tree®/d-tree™ Applications using Version 1.1 of the r-tree® Report Generator or Version 3.1 of the d-tree™ Development ToolBox will require updated versions of these development tools.

FAIRCOM
corporation

4006 WEST BROADWAY
COLUMBIA, MO 65203
PHONE 314.445.6833
800.234.8180
FAX 314.445.9698